基于 HBase 的列存储压缩策略的选择优化 *

孙靖超，芦天亮

(中国人民公安大学 信息技术与网络安全学院，北京 100076)

摘 要：大数据时代背景下，列存储数据库使用场景愈加增多，推动了列存储相关领域的研究进展。为解决现有列存储数据库压缩策略在压缩过程中遇到的数据离散程度大，分类粒度小，配套分类算法缺陷导致的学习成本高，压缩效率难以保证的问题，本文提出了一种基于排序的列区混合压缩策略，首先根据 HBase 特点设计了一种对各列数据进行排序的方法加强数据紧密度，然后根据数据特点分别使用混级区压缩策略和混级列压缩策略进行压缩策略推荐，在 TPC-DS 标准数据集上与前人策略进行比较，实验结果显示本文方法在压缩率、压缩/解压时间方面均有优异的表现，从而证明了本文方法的有效性。

关键词：列存储；数据压缩；HBase；压缩策略选择方法

中图分类号：TP　　doi: 10.3969/j.issn.1001-3695.2017.12.0841

Optimization of compression strategy selection method based on classification of HBase data

Sun Jingchao, Lu Tianliang

(*People's Public Security University of China*, *School of Information Technology & Network Security*, *Beijing* 100076, *China*)

Abstract: In the era of big data, the usage of column storage database is increasing, which promoted the development of research in column-oriented storage field. In order to solve the problem of high learning cost and low compression efficiency caused by large data dispersion, small classification granularity and the defect of applied classification algorithm encountered in the compression process of the existing column-based database compression strategy, this paper designed a sorted-based hybrid compression strategy of column-based compression and sector-based compression. Firstly, we designed a method to sort the data in each column according to the characteristics of HBase to strengthen the data compaction. Secondly, according to the characteristics of the data, we applied the hybrid column-based compression strategy and the hybrid sector-based compression strategy respectively to recommend the compression algorithm . We have conducted experiments on TPC-DS standard data and the results demonstrate that the proposed strategy has excellent performance in both compression rate and compression / decompression time.

Key words: column-oriented storage; data compression; HBase; selection method of compression strategy

0 引言

传统关系型数据库难以应付日益增多的非结构化数据和大数据带来的对灵活可扩展性的要求，为了弥补传统数据库的缺陷，非关系型数据库[1]应运而生。列存储数据库[2]是非关系型数据库中的重要一类，与传统数据库相比，在随机查询，多表关联，历史数据分析等具体应用场景中，处理速度能快几个数量级。HBase[3]是 Apache 的一个旨在将 Big Table 用于存储的开源项目，现如今已经成功运用在包括 Facebook,Twitter 和 Yahoo 等很多大型互联网公司的数据存储中，可以很好的适应大数据时代的存储工作。数据压缩是数据库存储系统常用的用来提升表现的方法，可以节省存储空间，使数据分布更密集从而减少搜索时间，增加数据传输速率和增加缓冲池命中率，减少 I/O 次数，从而缓解 cpu 和磁盘速度不匹配的问题，提高查询效率。HBase 中的数据按列进行存储，同一列的存储数据往往具有相似数据类型和数据特征，使得其与传统按行存储的数据库相比更适合于进行数据压缩。

J. Abadi 早在 06 年其对 C-store 配套压缩算法的研究文章里[4]就指出如何根据数据特点选择压缩算法会是下一步的研究重点，如今针对列存储数据库的特点进行压缩策略的选择的研究已经取得了很多成果。但前人工作在压缩时对数据并未进行预处理，数据在各部分分布差异较大，离散程度高，不适于

收稿日期：2017-12-08；**修回日期：**2018-01-23　**基金项目：**国家重点研发计划"网络空间安全"重点专项（2017YFB0802804）；国家自然科学基金项目（61602489）；赛尔网络下一代网络技术创新项目（NGII20160405）

作者简介：孙靖超（1993-），男，山东烟台人，硕士研究生，主要研究方法为大数据存储和网络安全（kenniyaligen@163.com）；芦天亮（1985-），副教授，主要研究方向为网络攻防，恶意代码.

压缩。在压缩粒度的选择上多偏好小粒度压缩策略，但小粒度策略要统计各区的统计信息，计算成本高，影响压缩时间，且研究对压缩算法本身未有足够关注。在压缩策略分类算法的选择上，前人工作中常采用的分类算法有决策树和朴素贝叶斯。决策树可解释性好，但在面对现实复杂的非结构化数据和各种噪声数据时，很容易过拟合。朴素贝叶斯分类方法有坚实的数学基础，算法简单，易于实现，但其分类性能受先验知识影响大，分类性能往往不尽人意[5]，导致策略的准确性和压缩效率无法保证，且现有应用的分类算法不支持并行处理，不能充分利用集群算力，使负载不均。

为解决现有列存储数据库中在压缩过程中遇到的数据分布离散程度大，以及现有分类粒度带来的复杂学习成本和现有关于压缩策略分类算法的缺点，对压缩算法的选择策略做了详细研究，并作出了如下贡献：

设计了一种在压缩前对数据进行排序的方法。将各列拆分，对拆分出的各列进行结构设计，使数据能够按照排序之后的顺序存储在各 region 中并且避免热点问题，按使数据能够排列紧密，最大程度减小数据在局部数据分布的差异性。

提出了一种列区混合压缩策略（a Hybrid Compression Strategy of Column-Based Compression and Sector-Based Compression）。根据列数据分布特性的不同适用不同压缩策略，对列中各区分布特征相似的列采用混级列压缩（Hybrid Column-Based Compression Strategy），数据特征不相似的采用混级区压缩（Hybrid Sector-Based Compression Strategy），大小粒度结合减少了学习成本。

在策略设计时，对压缩算法进行了详细研究，选定了合适于不同数据特点的压缩算法，并且首次在压缩策略中引入了 XGBoost 算法作为分类算法，其出色的泛化特性和对并行计算的支持弥补了以往分类算法的不足。

本文的其余部分安排如下：本文第二节介绍相关工作；第三节详细介绍了列级混合压缩算法的压缩策略选择方法；第四节用本文提出的方法在数据集上进行了实验，并对实验结果进行了详细的分析和说明；最后一部分进行了总结和展望。

1　相关工作

数据压缩一直是数据领域的重点关注问题，现存众多压缩方法，其中轻量级压缩方式有游程编码[6]，字典编码，空值抑制[7]等，重量级压缩编码有 GZIP，Lempel-Ziv 系列[8]，Huffman 编码[9]和算术编码[10]等。根据文献[11]，轻量级和重量级压缩算法的不同在于轻量级算法是对连续的值进行操作，重量级算法是打破了值间的边界，将值当成一系列字节进行操作。常用的轻量级和重量级压缩算法的分类如图 1 所示。

关于列存储压缩算法策略的研究首先开始于在对 C-store[13]的有关研究中，J. Abadi 等[4]在文章中提出了一种基于决策树的列压缩模型。该压缩模型通过建立了一个压缩算法决策树来判定各列的最佳压缩算法。但是该方法压缩粒度过大，忽视了数据的局部分布特点与整列数据的差异性对压缩带来的影响。

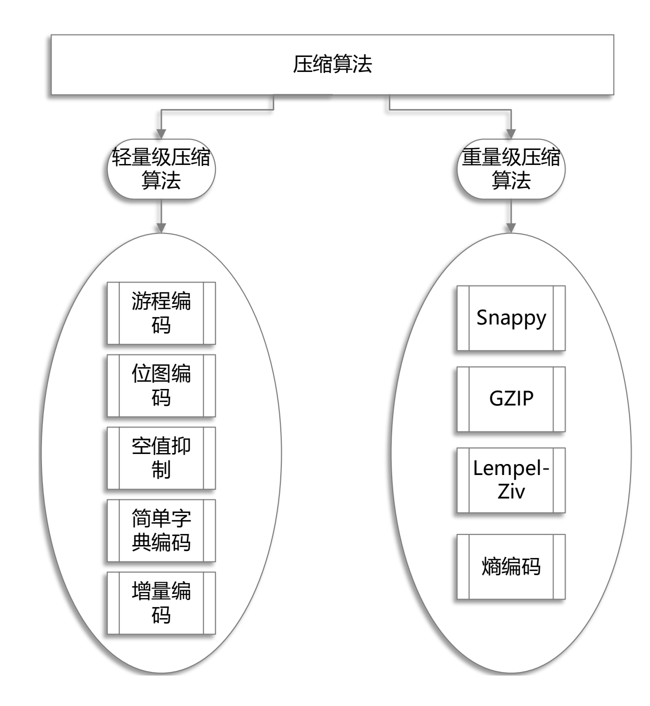


图 1　轻量级和重量级压缩算法

王振玺等人[14]提出了一种区级压缩策略，以区为单位将数据进行划分，以分区之间的相关性和差异性进行压缩算法的选择，找到适合本部分的压缩方法。此方法可以为不同区根据特点适用不同压缩算法，保证了压缩率，但是是如若相似度差别大的分区过多会导致较大计算量。

Idreos 等人[15]提出了一种基于贝叶斯分类的压缩算法动态选择策略，通过贝叶斯公式计算后为不同的数据块选择不同的压缩算法，尽可能使待压缩数据达到最佳的压缩效果。但是该方法的准确性很大程度取决于训练样本，且没有建立一个评估层根据反馈结果评估压缩算法的好坏。

王海艳等人[16]提出了一种基于冷热数据分类的压缩策略选择方法，首先依据数据文件访问频度将 HBase 数据划分为冷热数据，并改进了贝叶斯分类压缩方法，在前人基础上增加评估层，结合区级压缩策略的优势提出了一种新的压缩分类法。但其分类算法不支持并行处理，且在压缩粒度上并未有突破，区级学习的缺点仍然存在。

2　列区混合压缩算法的压缩策略选择方法

2.1　HBase 数据排序实现

以往关于列存储压缩策略的工作很少涉及关于数据排序的分类压缩策略，根据文献[17][18]，对数据按自身情况进行重新排序会使压缩算法得到更好的效果，达到更好的压缩率，占用更少的存储空间。

HBase 支持压缩的文件有 SequenceFile 和 HFile，其中 WAL（Write-Ahead Log）是 HBase 中主要的 SequenceFile，是一种预写日志。数据会首先写入 WAL，写入的 WAL 会存放到

MemStore 中，当存储已满，会刷写到 HDFS 中生成一个新的 HFile。在 HBase 中，各列的信息都存储在各 region 由 HFile 组成的 StoreFile 中，直接对原表内容是无法按内容进行排序存储的，因为各 region 是按照行健进行排序划分，每个 region 对应一定行健范围的内容，即相同行健的列存储在同一 region，而不论要对何种文件进行压缩，相邻行健的键值对总排列在一起，故如想实现对表内内容的排序，首先应将列拆为不同的表，对列值进行排序，对排序后的列生成新表的复合行健采用 $<columnID>_<rowID>_<Row-key>$ 格式规则进行设计。其中 $<columnID>$ 作为原列位置的标志，并且作为 salting 前缀使数据分布到不同的 region sever 上去，避免产生热点问题（hot spot）[3]，$<rowID>$ 的作用是使列中排序好的值能排列在一起，避免分散到不同的 region 中导致排序数据无法连续存储，$<Row-key>$ 的作用为记录原标标志，关联各列数据。其中 $<columnID>$，$<rowID>$ 此两字段的值应该定长。例如，有一 5 列 100 行的示例表，列族为 Cf，具体表格式如表 1 所示：

表 1　示例表

ROW-Key	Cf:ItemID	Cf:Country	Cf:Brand	Cf:Color	Cf:Price
1e9frt1	256331	China	Lining	Black	100$
2fvgao2	258221	Canada	Addidas	Purple	100$
1gvzf43	256893	American	Nike	White	100$
……	……	……	……	……	……

首先对数据按列排序，将排序后的数据存到新表中，新表复合行健采用 $<columnID>_<rowID>_<Row-key>$ 格式设计，列族仍然设为 Cf，这里以 ItemID 列为例展示为排序好的该列数据所建的表格式，具体表格式如表 2 所示:

表 2　ItemID 表结构

ROW-Key	Cf:ItemID
1_001_1e9frt1	255330
1_002_2fvgao2	255331
1_003_1gvzf43	255532
……	……

2.2　压缩算法和压缩粒度的选取

Abadi D[4]和 Ferreira[11]的研究表明在列存储应用场景中，轻量级压缩算法不仅 cpu 成本低，而且可以支持直接对压缩态数据的操作，提高查询效率。故压缩算法的选择原则就是偏重轻量级算法，兼顾重量级算法。

轻量级压缩算法中常用方法包括游程编码，位向量编码，空值抑制，简单字典编码和增量编码。在 HBase 中，对空值不进行存储，故不将空值抑制列入待选压缩算法。在行存储中，游程编码只能用于对连续的空格和字母的压缩，但在列存储中，游程编码的应用领域非常广泛，由于同一列的数据属性类似，经排序后又会有很适合的连续值长度，所以对基数小的列选用游程编码是非常合适的选择。在对数据的要求上，位向量算法对数据类型的要求较低，而游程编码对数据类型要求较高，但对于重复数据且排序规则的数据压缩效果很好，数据类型要求较高，两者优势互补，可以产生一个很好效果的算法。WAH（Word—Alignment Hyhrid Code）[19]算法很好的将两者进行结合，且在查询效率上有着比在未压缩的位图向量上更好的表现，故将其加入压缩算法集中。在真实数据中，经常会出现存储诸如 URL，家庭住址之类的非结构化数据，该类型数据用简单字典编码中的前缀编码（Trie Encoding）就可以达到很好的效果。而对日期，时间，或其他间距不大的数据类型，增量编码（Delta Encoding）是一个非常适合的轻量级压缩方法。

关于重量级算法，文献[3]将 GZIP、LZO、snappy 在 HBase 的表现进行了实验，LZO 的压缩率居中，但是其压缩/解压缩明显快的多，至于熵编码系列的 Huffman 编码和算术编码，文献[4]指出这两种编码方法在列存储数据库中的效果不好，不仅不支持压缩态的操作，且压缩/解压缩速度与上三种方法比没有任何优势。本文选取了文献[2]提出的一种改进的 LZO 方法加入压缩算法集，该方法与原 LZO 算法相比能节省最高 2 倍的存储空间和 10%的内存使用量，且解压速度比 snappy 更快。对那些数据分布不均匀，数据倾斜不明显的数据推荐用改进的 LZO 编码。

根据以上，最终选取了游程编码，位向量编码，WAH 编码，前缀编码，增量编码和改进的 LZO 作为压缩算法。

在列存储中以区为压缩粒度的目的主要是为了解决局部数据分布与整列数据分布不匹配的问题[22-24]，但是对于文献中指出的数据分布特点不同的问题，通过在压缩前对数据进行排序，可使数据排列更紧密，尤其对基数小，相同值所占百分比高，相同值连续平均数目多的列而言，各区之间差异很小，可直接以列为压缩粒度，故在本文中选择了以列区混合的方式作为压缩粒度。

2.3　基于 XGBoost 的压缩算法选择

XGBoost（eXtreme Gradient Boosting）[20]是 Tianqi Chen 等人提出的一种改进的梯度提升决策树（Gradient Boosted Decision Trees），因其快速的处理速度和优异的分类表现在短时间内便获得了广泛关注，在 Kaggle 等数据比赛中被广泛使用。拥有 GBDT 树的处理数据类型的灵活性，对异常值的鲁棒性，泛化能力强等优点，还在选择最佳分裂点时，进行并行枚举，解决了 GBDT 树无法并行训练的缺点，并且在设计时进行了充分的缓存线优化，加快了训练速度。支持列采样，构建树属性时进行采样，使训练效果快且好。采用正则化防止过拟合，进一步增强了泛化性能。

根据上文讨论，构建压缩算法集合 M ={游程编码，位向量编码，WAH 编码，前缀编码，增量编码，改进的 LZO，不压缩}，在构建的压缩集合中采用 XGBoost 来进行压缩算法的划分。

在用 XGBoost 做压缩分类时，每一次迭代需要有 m 棵树，m 是类别数目，每棵树对一个类别进行预测，在本例中 $m=7$。

可把每颗树看做一个函数 f，输入为统计量 T 时，f 将输入样本的统计量映射为 $f_1(T), f_2(T), f_3(T), ..., f_m(T)$ 做为 T 的预测值。

XGBoost 的流程重点在于建树过程和叶子节点分裂过程。

在建树过程中，最重要的就是目标函数，首先定义目标函数：

$$Obj(\phi) = L(\phi) + \Omega(\phi) \tag{1}$$

其中 $L(\phi)$ 是损失函数（cost function）。$L(\phi)$ 的表达式如下：

$$L(\phi) = \sum_i l(\hat{y}_i - y_i) \tag{2}$$

因为本文的压缩策略最终只选择一个，各类互斥，是一个多分类（multi-class）问题，而非一个多标签（multi-label）问题，故把损失函数定为 Softmax 损失函数，则属于某个类别 i 的概率为

$$p_i = e^{f_i(T)} / \sum_{m=1}^{M} e^{f_k(T)} \tag{3}$$

损失函数 $L(\phi)$ 中 $\hat{y}_i$ 的计算方式是

$$\hat{y}_i = \sum_{k=1}^{K} f_k(T) \tag{4}$$

K 是树的总数，f 表示每一个具体的 CART 树,模型由 k 棵 CART 树组成,

$$f_k(T) = \omega_q(T), \omega \in R^P, q: R^d \to \{1,2,3,....,P\} \tag{5}$$

R 指的是叶子权重，q 指的是树结构，q 把输入映射到叶子索引号上，ω 指定了每个索引号的叶子分数，其中 ω 的取值是通过函数最优化计算出的使目标函数最小的值。P 指的是索引总数。

$\Omega(\phi)$ 是正则化项,用来判断树的复杂程度。用以权衡目标函数的下降和模型的复杂程度,避免过拟合。

$$\Omega(\theta) = \gamma T_\theta + \frac{1}{2}\lambda \sum_{j=1}^{T_\theta} \omega_j^2 \tag{6}$$

T_θ 指叶子个数，ω 表示各叶子节点的值，其中 γ 值、λ 值指引入新叶子节点的复杂度代价，γ 值、λ 值值越大，对有较多叶子节点和极端值的树惩罚越大，$\Omega(\phi)$ 的取值越小树结构越简单。

每一棵树结构的确定是通过每次尝试对一个已有的叶子加入一个分割来实现的，XGBoost 对树节点分裂采用的公式为

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \Upsilon \tag{7}$$

其中 $\frac{G_L^2}{H_L + \lambda}$ 指左子树分数，$\frac{G_R^2}{H_R + \lambda}$ 指右子树分数，$\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}$ 指不分割可以得到的分数，Υ 指加入新节点引入的复杂度代价，G_L 是左子树所有节点一阶导数的和，G_R 是右子树所有导数一阶的和，H_L 是左子树所有节点二阶导数的和，H_R 是右子树所有节点二阶导数的和。XGBoost 的节点分裂操作和普通的决策树分裂过程不同在于普通的决策树在分裂的时候并不考虑树的复杂度，而依赖后续，的剪枝，而 XGBoost 在分裂的时候就已经通过 Υ 考虑了树的复杂度，故不需要进行单独的剪枝操作。当分裂带来的增益小于阀值 Υ 时，停止分裂。然后进入下一轮迭代，当样本权重和小于设定阈值时则停止建树。

2.4 列区混合压缩策略流程

本文压缩策略的设计原则如下：

1.尽量减少计算成本，2.尽量提高压缩效率。

根据设计原则设计如下方案，

1.要尽量减少计算成本，就得从压缩粒度和压缩算法上着手，压缩粒度要混合列级和区级压缩，不能单纯利用区级压缩，为了避免压缩粒度选择过大导致压缩算法在局部不适用的情况，先采用排序手段使数据排序紧密。在压缩算法的选择上也应该偏重选用轻量级压缩。2. 要保证最高的压缩效率，就需要针对不同的数据特点设计不同的压缩算法，并且改进以往分类策略中分类算法的缺陷以达到最佳压缩效率。根据方案思路，设计了如下基于排序的列区混合压缩策略（a Hybrid Compression Strategy of Column-Based Compression and Sector-Based Compression）。

在进行压缩策略的详细介绍时，首先介绍相关定义。

定义 1 数据特征：数据特征是指用来对所选数据进行描述的一组信息。这组信息包括基数 q，相同值的总数 a ,数据类型 t，数据的倾斜程度 d，键值对总数 v ,相同值连续的平均数目 c，值的平均长度 l。在本文中基数特指一列中数据的散列程度，即该种不同属性值的个数。数据访问频度 $f = C/t$，C 指文件访问次数，t 指相应时间段。

定义 2 数据统计量 T：数据统计量是通过数据特征的到的一组数据，是用来作为压缩策略的输入，通过数据特征计算得到，共有 7 个分量。分别是 q_1 数据的散列程度，q_2 数据的倾斜程度，q_3 相同值所占百分比 a*100/v，q_4 相同值连续的平均数目 c，q_5 数据类型 t，q_6 值的平均长度 l，q_7 数据使用频度 f。

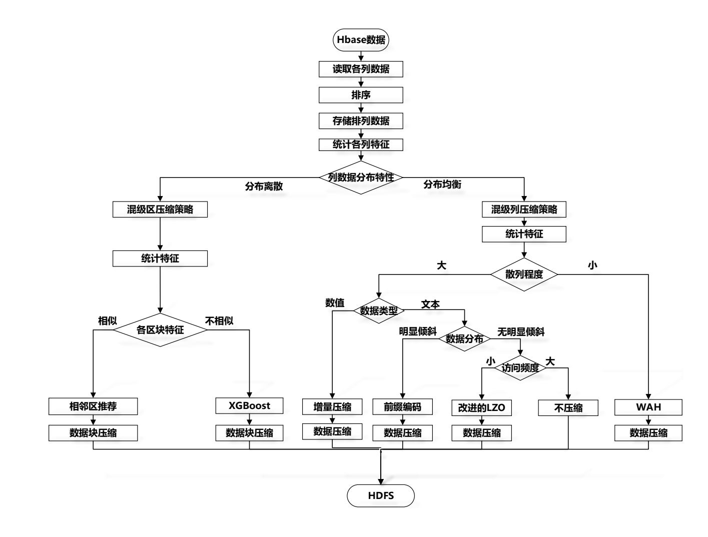


图 2　基于排序的列区混合压缩策略流程图

基于排序的列区混合压缩策略（a Hybrid Compression Strategy of Column-Based Compression and Sector-Based

Compression）流程如图 2 所示。其算法流程描述如下。

算法 1：基于排序的列区混合压缩策略

输入：待压缩数据 W

输出：压缩是否成功（0：失败，1：成功）

$Step1$ 从 HBase 读入各列数据。

$Step2$ 对各列数据进行排序,并对各列数据按照指定格式进行存储。

$Step3$ 随机抽取列中 10 区统计特征统计量 $T_i=\{q_2,q_3,q_4,q_5,q_6,q_7\}$, $i\in[1,10]$ 。

$Step4$ 判断各列数据分布特性，根据数据分布特性将数据用混级区压缩策略（Hybrid Sector-Based Compression）和混级列压缩策略（Hybrid Column-Based Compression）分别进行存储，

$Step5$ 各列数据根据分配的不同压缩策略进行压缩。

$Step6$ 将压缩数据存储到 HDFS 中。

首先从 HBase 读入各列数据，对各列数据进行排序，按照 3.1 部分提出的表结构对排序后的数据进行存储。通过统计随机块的统计量来计算各区之间的相似因子 S ，相似因子 S 是用来判断区间相似度的定义量。通过两区的统计量 T 特点分量的差的绝对值得到。其中需要的统计分量是通过第一个区的压缩策略给出的，如若选出的列中各区特征相似程度高，则判断列分布均衡，列中数据适用混级列压缩策略，若选出的列中各区特征相似程度低，则判断分布离散，列中数据适用混级区压缩策略。然后用适用策略根据数据特点判断出的压缩算法对数据进行压缩后将数据存储到 HDFS 中。在下面详细介绍了混级区压缩策略和混级列压缩策略的算法流程。

算法 2：混级区压缩策略（Hybrid Sector-Based Compression Strategy）

输入：待压缩列数据

输出：压缩策略向量 M_s

$Step1$ 令 $i=1$。

$Step2$ 统计 $T_i=\{q_1,q_2,q_3,q_4,q_5,q_6\}$ 。

$Step3$ 若 $i=1$,跳转 $Step4$,否则跳转 $Step3$ 。

$Step4$ 通过相似因子 S 计算与上一区块的相似度,若相似度高, $m_i=m_{i-1}$,否则跳转统计 $T_i=\{q_1,q_2,q_3,q_4,q_5,q_6\}$,跳转到 $Step5$ 。

$Step5$ 对数据块使用基于 XGBoost 的策略选择方法。

$Step6$ 如块 i 不是最后一个块, $i=i+1$,跳到 $Step3$ 。

$Step7$ 返回压缩策略向量 M_s

对适用混级区压缩策略的列首先统计第一个区块的统计量 $T_i=\{q_1,q_2,q_3,q_4,q_5,q_6\}$ ， $i=1$。 通过相似因子 S 的计算判断两区的相似性，若两区相似，则适用上一区的压缩算法，否则重新统计统计量 $T_i=\{q_1,q_2,q_3,q_4,q_5,q_6\}$ ，利用 XGBoost 策略得到该区块相应的压缩算法。将每个区块根据 XGBoost 策略或是相邻区学习策略得到的压缩策略存入压缩策略向量 M_s 。

算法 3：混级列压缩策略（Hybrid Column-Based Compression Strategy）

输入：待压缩列数据

输出：压缩策略 m

$Step1$ 统计特征统计量 $T_c=\{q_1,q_2,q_5,q_6,q_7\}$

$Step2$ 对基数 q 进行判断,如小于阈值, m =WAH 编码,跳转 $Step6$,如大于阈值,跳转 $Step3$ 。

$Step3$ 判断文本类型 t ,如是数值, m =增量压缩编码,跳转 $Step6$,如是文本,跳转 $Step4$ 。

$Step4$ 判断数据倾斜,如数据有明显倾斜, m =前缀编码,跳转 $Step6$,如无明显倾斜,跳转 $Step5$ 。

$Step5$ 根据使用频度 l ，将压缩算法分为改进的 LZO 和不压缩，若使用频度大 m =不压缩，跳转最后一步，若使用频度小， m =改进的 LZO，跳转 $Step6$ 。

$Step6$ 返回压缩策略 m 。

首先第一步对读入的数据进行排序，本文选择了以字典序为排序方式，第二步对排好序的数据进行基数的判断，之所以将基数作为首要分类标准是因为 WAH 算法在权衡压缩率，压缩/解压时间和查询效率的表现后在四种算法中是最优秀的，所以希望尽可能多的数据适用 WAH 算法，而增量压缩的数据适应类型较窄，对数据要求较高，所以安排在第三步进行判断，最后第四步对剩下的数据根据数据分布特点进行分类，对有明显倾斜的数据适用前缀压缩，对无明显倾斜的数据根据使用频度分别选择改进的 LZO 算法或是不进行数据压缩。

3 实验验证

为了检验本文提出的压缩策略的效果，选取了文献[4]中的 c-store 配套压缩策略,文献[14]中基于学习的区级压缩策略，文献[15]提出的朴素贝叶斯理论的压缩策略选择方法，和文献[16]提出的基于冷热数据的压缩分类策略和本文方法对相同的数据进行压缩，通过压较缩率和压缩/解压时间两个指标来验证本文方法的可行性。

1）实验环境

本实验环境采用 9 台系统为 centos7 的服务器，其中一台 MASTER，8 台 SLAVER，服务器硬件配置相同，cpu 均为 Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz，64gb 内存，3t 硬盘，hadoop 版本 2.7.3，hbase 版本 1.2.4。

2）数据集描述

关于列式数据的压缩，前人采用的 TPC-H 数据集主要适用关系数据 OLAP 的操作，且数据不倾斜，数据形式单一，与真实的使用场景差距较大。本文采用的的是 TPC-DS 数据集 ，是权威标准化组织 TPC（Transaction Processing Performance Council）根据 TPC-H 的不足和现实场景的需求改进了的版本，测试用的数据有倾斜，和现实数据较为一致。

本实验选用了其中的 ITEM 表作为测试数据集，这是因为该表数据类型丰富，符合实际且更能体现出各压缩算法的性能，用选定的策略对该表进行压缩，该表共有 22 种属性，本实验通过 dsdgen 程序生成了 3414MB 大小的 ITEM 表。生成数据的各

列文件访问级别表如表 3 所示。

表 3 各列对应的文件访问级别

列	访问级别	列	访问级别
1	沉默数据	12	沉默数据
2	沉默数据	13	沉默数据
3	沉默数据	14	沉默数据
4	沉默数据	15	沉默数据
5	沉默数据	16	沉默数据
6	沉默数据	17	活泼数据
7	活泼数据	18	沉默数据
8	沉默数据	19	沉默数据
9	沉默数据	20	活泼数据
10	沉默数据	21	活泼数据
11	活泼数据	22	活泼数据

3）实验结果与分析

实验在选定数据上将各压缩策略在压缩率和压缩/解压时间方面进行了比较，结果如图 3～图 6 所示。

a）压缩率：

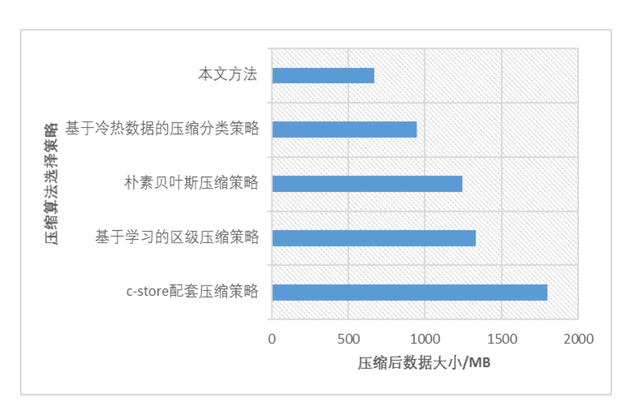


图 3 各策略压缩率比较

由图 3 可知，本文提出的方法在各列的压缩效果比其他 4 种策略都要好，压缩率达到了原数据的 20%左右，基于冷热数据的压缩配套策略次之，压缩率为 27.8%，朴素贝叶斯压缩策略的压缩率为 36.5%，基于学习的区级压缩策略为 38.9%，c-store 配套压缩策略的压缩率最低为 52.8%，本文取得的压缩效果较好是因为以前的策略大多将研究重点放在分类算法上，而对压缩算法的选择未有足够关注，多是沿用了 c-store 配套策略的压缩算法。采取细粒度的分类虽然能提高对数据的分类精度，但是如果压缩算法不够合适，再高的分类精度也无法达到很好的准确率。本文算法的选择充分考虑了大数据环境下各类数据的特点，针对不同特点的数据选择了适合的压缩算法。且在压缩前对数据进行排序，各部分数据特点清晰，便于压缩，能达到很高的压缩比，具体各列压缩效果如图 4 所示。

由图 4 可知，本文方法在除 17 列外各列的压缩效果都比其他方法要好，其中 17 列 i_informulaiton 列属于活泼数据，根据本文策略判断数据特点后断定为不被压缩，但是基于区级的压缩策略粒度更小，某些文本可能局部有相似性，所以被判断使用压缩算法，可以看到，其他压缩算法在该列效果并不好，而且虽然能够起到压缩效果，但是会耗费较多的压缩时间，而且对后续查询造成困难。

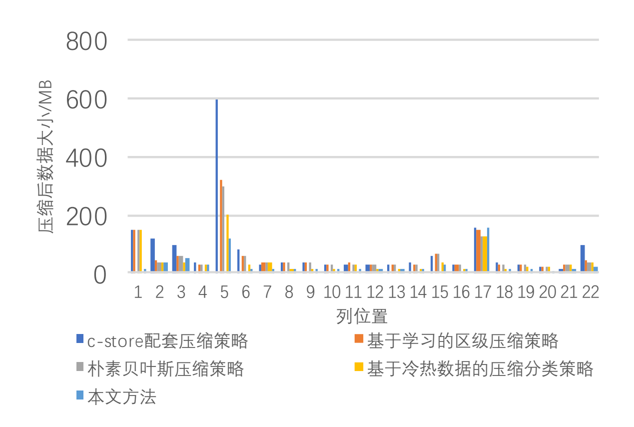


图 4 各策略在不同列的压缩效果

b）压缩/解压时间：

压缩/解压时间的实验结果如图 5、图 6 所示。

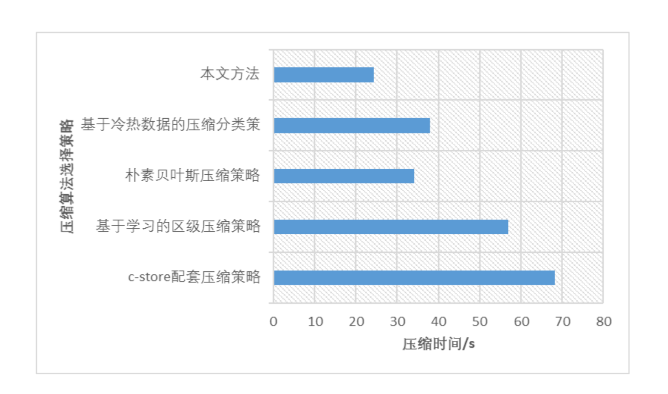


图 5 压缩时间

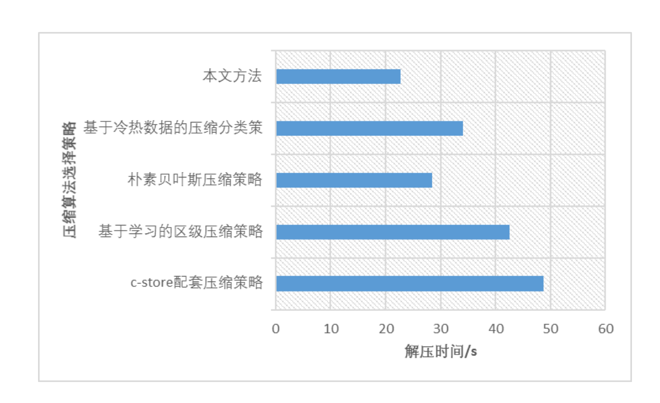


图 6 解压时间

结果证明本文提出的策略在压缩/解压时间方面也有很好的优势，这是因为本文方法通过混合粒度压缩，将列根据不同特点分别采用不同压缩粒度，不需要向以往工作那样对列中所有区的数据进行统计，节省了大量的计算成本。而且在压缩前对数据进行排序，除了有利于压缩比的同时，更紧凑严密的数据分布可以节省大量轻量级算法压缩的时间成本，以 WAH 算

法为例，数据排序后，有相同特征的值排在一起，可以增加游程长度，减少三元组数量，从而加快压缩时间和解压时间。

4 结束语

本文根据前人对列存储压缩策略工作的不足，针对 HBase 的特点，提出了一种压缩前对列数据排序的方法；在此基础上，将按区和按列两个压缩粒度结合，提出了一种列区混合的压缩策略，在两种压缩粒度下分别提出了适应排序后数据特点的压缩策略；对压缩算法进行了详细研究，引入了 XGBoost 算法来解决以往压缩分类算法的不足。实验证明，本文方法在压缩率，压缩/解压时间上均有优于前人的表现。接下来的研究重点是基于 HBase 的数据库压缩态数据查询的优化上，使压缩数据在 HBase 中不仅有存储优势，更有查询优势。

参考文献：

[1] Cattell, Rick. Scalable SQL and NoSQL data stores [J]. Acm Sigmod Record, 2011, 39 (4): 12-27.

[2] Hall A, Bachmann O, Büssow R, *et al.* [J]. Proceedings of the Vldb Endowment, 2012, 5 (11) .

[3] George L. HBase: the definitive guide [J]. Andre, 2015, 12 (1): 1-4.

[4] Abadi D, Madden S, Ferreira M. Integrating compression and execution in column-oriented database systems [C]// ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, Usa, June. DBLP, 2006: 671-682.

[5] Caruana R, Niculescu-Mizil A. An empirical comparison of supervised learning algorithms [C]// International Conference on Machine Learning. ACM, 2006: 161-168.

[6] Tanaka H, Leon-Garcia A. Efficient run-length encodings [J]. IEEE Transactions on Information Theory, 1982, 28 (6): 880-890.

[7] Roth M A, Horn S J V. Database compression [J]. Acm Sigmod Record, 1993, 22 (3): 31-39.

[8] Ziv J, Lempel A. A Universal Algorithm for Sequential Data Compression [J]. IEEE Transactions on Information Theory 23 (3) , 337-343.

[9] Huffman D. A method for the construction of minimum-redundancy codes [J]. Resonance, 2006, 11 (2): 91-99.

[10] Howard P G, Vitter J S. Arithmetic Coding for Data Compression [M]. Springer New York, 2016.

[11] Ferreira M C. Compression and query execution within column oriented databases [J]. Massachusetts Institute of Technology, 2005.

[12] Boncz P A, Manegold S, Kersten M L. Database Architecture Optimized for the New Bottleneck: Memory Access [C]// International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. 1999: 54-65.

[13] Stonebraker M, Abadi D J, Batkin A, *et al.* C-store: a column-oriented DBMS [C]// International Conference on Very Large Data Bases, Trondheim, Norway, August 30-September. DBLP, 2005: 553-564.

[14] 王振玺，乐嘉锦，王梅，等. 列存储数据区级压缩模式与压缩策略选择方法 [J]. 计算机学报, 2010, 33 (8): 001523-1530.

[15] Idreos S, Kersten M L, Manegold S. Self-organizing tuple reconstruction in column-stores [C]// ACM SIGMOD International Conference on Management of Data. ACM, 2009: 297-308.

[16] 王海艳，伏彩航. 基于 HBase 数据分类的压缩策略选择方法 [J]. 通信学报, 2016, 37 (4): 12-22.

[17] Lemire D, Kaser O, Gutarra E. Reordering rows for better compression: Beyond the lexicographic order [J]. Acm Transactions on Database Systems, 2012, 37 (3): 1-29.

[18] Lemire D, Kaser O. Reordering columns for smaller indexes [M]. Elsevier Science Inc. 2011.

[19] Wu K, Otoo E J, Shoshani A. Notes on Design and Implementation of Compressed Bit Vectors [J]. 2013.

[20] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System [J]. 2016: 785-794.